Metagenomics Workshop Documentation

Release 1

Alex Sczyrba

Aug 16, 2017

Contents

Welcome to the one-day metagenomics assembly workshop. This tutorial will guide you through the typical steps of metagenome assembly and binning.

CHAPTER 1

The Tutorial Data Set

Note: This tutorial was prepared for a training workshop utilizing our local compute infrastrucutre. If you want to download the data set to your machine and run it locally, you can find the data here.

We have prepared a small toy data set for this tutorial. The data is located in */vol/metagencourse/DATA/WGS-data* directory, which has the following content:

File	Content
genomes/	Directory containing the reference genomes
gold_std/	Gold Standard assemblies
read1.fq	Read 1 of paired reads (FASTQ)
read2.fq	Read 2 of paired reads (FASTQ)
reads.fas	Shuffled reads (FASTA)

Create a working directory in your home directory and symbolic links to the data files:

```
cd ~/workdir
mkdir assembly
cd assembly
ln -s /vol/metagencourse/DATA/WGS-data/* .
```

CHAPTER 2

FastQC Quality Control

FastQC aims to provide a simple way to do some quality control checks on raw sequence data coming from high throughput sequencing pipelines. It provides a modular set of analyses which you can use to give a quick impression of whether your data has any problems of which you should be aware before doing any further analysis.

The main functions of FastQC are

- Import of data from BAM, SAM or FastQ files (any variant)
- Providing a quick overview to tell you in which areas there may be problems
- Summary graphs and tables to quickly assess your data
- Export of results to an HTML based permanent report
- Offline operation to allow automated generation of reports without running the interactive application

See the FastQC home page for more info.

To run `FastQC` on our data, simply type:

```
cd ~/workdir/assembly
fastqc
```

Start the analysis by loading the FASTQ files using menu "File -> Open..."

Check out the FastQC home page for examples of reports including bad data.

CHAPTER 3

Assembly

We are going to use different assemblers and compare the results.

Velvet Assembly

Velvet was one of the first de novo genomic assemblers specially designed for short read sequencing technologies. It was developed by Daniel Zerbino and Ewan Birney at the European Bioinformatics Institute (EMBL-EBI). Velvet currently takes in short read sequences, removes errors then produces high quality unique contigs. It then uses paired-end read and long read information, when available, to retrieve the repeated areas between contigs. See the Velvet home page for more info.

Step 1: velveth

velveth takes in a number of sequence files, produces a hashtable, then outputs two files in an output directory (creating it if necessary), Sequences and Roadmaps, which are necessary for running velvetg in the next step.

Let's create multiple hashtables using kmer-lengths of 31 and 51. We are going to submit the jobs to the compute cluster using qsub,asking for a machine with 24 cores (-pe multislot 24). You can check the status of your job using the command qstat:

```
cd ~/workdir/assembly/

qsub -cwd -pe multislot 24 -N velveth_31 -l mtc=1 -b y \
/vol/cmg/bin/velveth velvet_31 31 -shortPaired -fastq -separate read1.fq read2.fq

qsub -cwd -pe multislot 24 -N velveth_51 -l mtc=1 -b y \
/vol/cmg/bin/velveth velvet_51 51 -shortPaired -fastq -separate read1.fq read2.fq
```

Note: You can check the status of your job using the command qstat:

```
>>>statler:~/workdir/assembly>qstat
job-ID  prior   name       user         state submit/start at     queue                  ␣
↪           slots ja-task-ID
```

```
--------------------------------------------------------------------------------------
↪---------------------------
3888958 0.65000 velveth_31 asczyrba     r     11/21/2015 07:18:29 all.q@suc01016.
↪CeBiTec.Uni-Bie    24
3888959 0.45213 velveth_51 asczyrba     r     11/21/2015 07:18:45 all.q@suc01003.
↪CeBiTec.Uni-Bie    24
```

If you do not see your jobs using `qstat` anymore, they are finished. You should have two output directories for the two different kmer-lengths: *velvet_31* and *velvet_51*.

Step 2: velvetg

Now we have to start the actual assembly using `velvetg`. `velvetg` is the core of Velvet where the de Bruijn graph is built then manipulated. Let's run assemblies for both kmer-lengths. See the Velvet manual for more info about parameter settings. Again, we submit the job to the compute cluster:

```
qsub -cwd -pe multislot 24 -N velvetg_31 -l mtc=1 -b y \
/vol/cmg/bin/velvetg velvet_31 -cov_cutoff auto -ins_length 270 -min_contig_lgth 500 -
↪exp_cov auto

qsub -cwd -pe multislot 24 -N velvetg_51 -l mtc=1 -b y \
/vol/cmg/bin/velvetg velvet_51 -cov_cutoff auto -ins_length 270 -min_contig_lgth 500 -
↪exp_cov auto
```

The contig sequences are located in the *velvet_31* and *velvet_51* directories in file *contigs.fa*. Let's get some very basic statistics on the contigs. The script `getN50.pl` reads the contig file and computes the total length of the assembly, number of contigs, N50 and largest contig size. In our example we will exclude contigs shorter than 500bp (option *-s 500*):

```
getN50.pl -s 500 -f velvet_31/contigs.fa
getN50.pl -s 500 -f velvet_51/contigs.fa
```

Note: Most jobs above will be started on the compute cluster using the `qsub`.

- `qstat`: check the status and JOBNUMBER of your jobs
- `qdel JOBNUMBER`: delete job with job number JOBNUMBER

We usually submit the jobs to the cluster giving them a job name by using `-N JOBNAME`. This will create log-files named

- `JOBNAME.oJOBNUMBER`: standard output messages of the tool
- `JOBNAME.eJOBNUMBER`: standard error messages of the tool

You can look into these files by typing e.g. `less JOBNAME.oJOBNUMBER` (hit `q` to quit) or `tail -f JOBNAME.oJOBNUMBER` (hit `^C` to quit).

MEGAHIT Assembly

MEGAHIT is a single node assembler for large and complex metagenomics NGS reads, such as soil. It makes use of succinct de Bruijn graph (SdBG) to achieve low memory assembly. MEGAHIT can optionally utilize a CUDA-enabled GPU to accelerate its SdBG contstruction. See the MEGAHIT home page for more info.

MEGAHIT can be run by the following command. As our compute instance have multiple cores, we use the option *-t 24* to tell MEGAHIT it should use 24 parallel threads. The output will be redirected to file *megahit.log*:

```
cd ~/workdir/assembly/

qsub -cwd -pe multislot 24 -N megahit -l mtc=1 -b y \
/vol/cmg/bin/megahit -1 read1.fq -2 read2.fq -t 24 -o megahit_out
```

The contig sequences are located in the *megahit_out* directory in file *final.contigs.fa*. Again, let's get some basic statistics on the contigs:

```
getN50.pl -s 500 -f megahit_out/final.contigs.fa
```

Note: Most jobs above will be started on the compute cluster using the `qsub`.

- `qstat`: check the status and JOBNUMBER of your jobs
- `qdel JOBNUMBER`: delete job with job number JOBNUMBER

We usually submit the jobs to the cluster giving them a job name by using `-N JOBNAME`. This will create log-files named

- `JOBNAME.oJOBNUMBER`: standard output messages of the tool
- `JOBNAME.eJOBNUMBER`: standard error messages of the tool

You can look into these files by typing e.g. `less JOBNAME.oJOBNUMBER` (hit `q` to quit) or `tail -f JOBNAME.oJOBNUMBER` (hit `^C` to quit).

IDBA-UD Assembly

IDBA is the basic iterative de Bruijn graph assembler for second-generation sequencing reads. IDBA-UD, an extension of IDBA, is designed to utilize paired-end reads to assemble low-depth regions and use progressive depth on contigs to reduce errors in high-depth regions. It is a generic purpose assembler and epspacially good for single-cell and metagenomic sequencing data. See the IDBA home page for more info.

IDBA-UD requires paired-end reads stored in single FastA file and a pair of reads is in consecutive two lines. You can use *fq2fa* (part of the IDBA repository) to merge two FastQ read files to a single file. The following command will generate a FASTA formatted file called *reads12.fas* by "shuffling" the reads from FASTQ files *read1.fq* and *read2.fq*:

```
cd ~/workdir/assembly/

qsub -cwd -N fq2fa -l mtc=1 -b y \
/vol/cmg/bin/fq2fa --merge read1.fq read2.fq reads12.fas
```

IDBA-UD can be run by the following command. As our compute instances have multiple cores, we use the option *–num_threads 24* to tell IDBA-UD it should use 24 parallel threads:

```
cd ~/workdir/assembly/

qsub -cwd -pe multislot 24 -N idba_ud -l mtc=1 -b y \
/vol/cmg/bin/idba_ud -r reads12.fas --num_threads 24 -o idba_ud_out
```

The contig sequences are located in the *idba_ud_out* directory in file *contig.fa*. Again, let's get some basic statistics on the contigs:

```
getN50.pl -s 500 -f idba_ud_out/contig.fa
```

Note: Most jobs above will be started on the compute cluster using the `qsub`.

- `qstat`: check the status and JOBNUMBER of your jobs
- `qdel JOBNUMBER`: delete job with job number JOBNUMBER

We usually submit the jobs to the cluster giving them a job name by using `-N JOBNAME`. This will create log-files named

- `JOBNAME.oJOBNUMBER`: standard output messages of the tool
- `JOBNAME.eJOBNUMBER`: standard error messages of the tool

You can look into these files by typing e.g. `less JOBNAME.oJOBNUMBER` (hit `q` to quit) or `tail -f JOBNAME.oJOBNUMBER` (hit `^C` to quit).

Ray Assembly

Ray is a parallel software that computes de novo genome assemblies with next-generation sequencing data. Ray is written in C++ and can run in parallel on numerous interconnected computers using the message-passing interface (MPI) standard. See the Ray home page for more info.

Ray can be run by the following command using a kmer-length of 31. As our compute instance have multiple cores, we specify this in the ‘mpiexec -n 48 ‘ command to let Ray know it should use 48 parallel MPI processes:

```
cd ~/workdir/assembly/

qsub -cwd -pe multislot 48 -N ray -l mtc=1 -b y \
/usr/lib64/openmpi/bin/mpiexec -n 48 /vol/cmg/bin/Ray -k 31 -p read1.fq read2.fq -o␣
↪ray_31
```

This will create the output directory *ray_31* and the final contigs are located in *ray_31/Contigs.fasta*. Again, let’s get some basic statistics on the contigs:

```
getN50.pl -s 500 -f ray_31/Contigs.fasta
```

Now that you have run assemblies using Velvet, MEGAHIT, IDBA-UD and Ray, let’s have a quick look at the assembly statistics of all of them:

```
cd ~/workdir/assembly/
/vol/metagencourse/bin/get_assembly_stats.sh
```

Note: Most jobs above will be started on the compute cluster using the `qsub`.

- `qstat`: check the status and JOBNUMBER of your jobs
- `qdel JOBNUMBER`: delete job with job number JOBNUMBER

We usually submit the jobs to the cluster giving them a job name by using `-N JOBNAME`. This will create log-files named

- `JOBNAME.oJOBNUMBER`: standard output messages of the tool
- `JOBNAME.eJOBNUMBER`: standard error messages of the tool

You can look into these files by typing e.g. `less JOBNAME.oJOBNUMBER` (hit `q` to quit) or `tail -f JOBNAME.oJOBNUMBER` (hit `^C` to quit).

CHAPTER 4

Gene Prediction

Prodigal (Prokaryotic Dynamic Programming Genefinding Algorithm) is a microbial (bacterial and archaeal) gene finding program developed at Oak Ridge National Laboratory and the University of Tennessee. See the Prodigal home page for more info.

To run `prodigal` on our data, simply type:

```
cd ~/workdir/assembly/megahit_out

qsub -cwd -N prodigal -l mtc=1 -b y \
/vol/biotools/bin/prodigal -p meta -a final.contigs.genes.faa -d final.contigs.genes.
↪fna -f gff -o final.contigs.genes.gff -i final.contigs.fa
```

Output files:

final.contigs.genes.gff	positions of predicted genes in GFF format
final.contigs.genes.faa	protein translations of predicted genes
final.contigs.genes.fna	nucleotide sequences of predicted genes

CHAPTER 5

Assembly Evaluation

We are going to evaluate our assemblies using the reference genomes.

Read Mapping

In this part of the tutorial we will look at the assemblies by mapping the reads to the assembled contigs. Different tools exists for mapping reads to genomic sequences such as bowtie or bwa. Today, we will use the tool BBMap.

BBMap: Short read aligner for DNA and RNA-seq data. Capable of handling arbitrarily large genomes with millions of scaffolds. Handles Illumina, PacBio, 454, and other reads; very high sensitivity and tolerant of errors and numerous large indels. Very fast. See the BBMap home page for more info.

`bbmap` needs to build an index for the contigs sequences before it can map the reads onto them. Here is an example command line for mapping the reads back to the MEGAHIT assembly:

```
cd ~/workdir/assembly/megahit_out

qsub -cwd -N bbmap_index -l mtc=1 -b y \
/vol/cmg/bin/bbmap.sh ref=final.contigs.fa
```

Now that we have an index, we can map the reads:

```
qsub -cwd -pe multislot 24 -N bbmap -l mtc=1 -b y \
/vol/cmg/bin/bbmap.sh in=../read1.fq in2=../read2.fq out=megahit.sam␣
↪bamscript=sam2bam.sh threads=24
```

`bbmap` produces output in SAM format by default, usually you want to convert this into a sorted BAM file. `bbmap` creates a shell script which can be used to convert `bbmap`'s output into BAM format:

```
qsub -cwd -pe multislot 4 -N bbmap_sam2bam -l mtc=1 sam2bam.sh
```

SAM and BAM files can be viewed and manipulated with SAMtools. Let's first build an index for the FASTA file:

```
samtools faidx final.contigs.fa
```

To look at the BAM file use:

```
samtools view megahit_sorted.bam | less
```

We will use IGV: Integrative Genomics Viewer to look at the mappings:

```
cd ~/workdir/assembly/megahit_out
igv.sh
```

Now let's look at the mapped reads:

1. Load the contig sequences into IGV. Use the menu `Genomes->Load Genome from File...`
2. Load the BAM file into IGV. Use menu `File->Load from File...`
3. Load the predicted genes as another track. Use menu `File->Load from File...` to load the GFF file.

MetaQUAST

QUAST stands for QUality ASsessment Tool. The tool evaluates genome assemblies by computing various metrics. You can find all project news and the latest version of the tool at sourceforge. QUAST utilizes MUMmer, GeneMarkS, GeneMark-ES, GlimmerHMM, and GAGE. In addition, MetaQUAST uses MetaGeneMark, Krona tools, BLAST, and SILVA 16S rRNA database. See the QUAST home page for more info.

To call `metaquast.py` we have to provide reference genomes which are used to calculate a number of different metrics for evaluation of the assembly. In real-world metagenomics, these references are usually not available, of course:

```
cd ~/workdir/assembly

qsub -cwd -pe multislot 24 -N metaquast -l mtc=1 -b y \
/vol/cmg/bin/metaquast.py --threads 24 --gene-finding --meta \
-R /vol/metagencourse/DATA/WGS-data/genomes/Aquifex_aeolicus_VF5.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Bdellovibrio_bacteriovorus_HD100.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Chlamydia_psittaci_MN.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Chlamydophila_pneumoniae_CWL029.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Chlamydophila_pneumoniae_J138.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Chlamydophila_pneumoniae_LPCoLN.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Chlamydophila_pneumoniae_TW_183.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Chlamydophila_psittaci_C19_98.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Finegoldia_magna_ATCC_29328.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Fusobacterium_nucleatum_ATCC_25586.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Helicobacter_pylori_26695.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Lawsonia_intracellularis_PHE_MN1_00.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Mycobacterium_leprae_TN.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Porphyromonas_gingivalis_W83.fna,\
/vol/metagencourse/DATA/WGS-data/genomes/Wigglesworthia_glossinidia.fna \
-o quast \
-l MegaHit,Ray_31,velvet_31,velvet_51,idba_ud \
megahit_out/final.contigs.fa \
ray_31/Contigs.fasta \
velvet_31/contigs.fa \
velvet_51/contigs.fa \
idba_ud_out/contig.fa
```

QUAST generates HTML reports including a number of interactive graphics. You can load the reports into your web browser:

```
firefox quast/summary/report.html
firefox quast/combined_quast_output/report.html
```

CHAPTER 6

Binning

After the assembly of metagenomic sequencing reads into contigs, binning algorithms try to recover individual genomes to allow access to uncultivated microbial populations that may have important roles in the samples community.

MaxBin Binning

MaxBin is a software that is capable of clustering metagenomic contigs into different bins, each consists of contigs from one species. MaxBin uses the nucleotide composition information and contig abundance information to do achieve binning through an Expectation-Maximization algorithm. For users’ convenience MaxBin will report genome-related statistics, including estimated completeness, GC content and genome size in the binning summary page. See the MaxBin home page for more info.

Let’s run a MaxBin binning on the MEGAHIT assembly. First, we need to generate an abundance file from the mappes reads:

```
cd ~/workdir/assembly/megahit_out
pileup.sh in=megahit.sam  out=cov.txt
awk '{print $1"\t"$5}' cov.txt | grep -v '^#' > abundance.txt
```

Next, we can run MaxBin:

```
qsub -cwd -pe multislot 24 -N maxbin -l mtc=1 -b y \
/vol/cmg/lib/MaxBin-2.1.1/run_MaxBin.pl -thread 24 -contig final.contigs.fa -out␣
↪maxbin -abund abundance.txt
```

Assume your output file prefix is (out). MaxBin will generate information using this file header as follows.

(out).0XX.fasta	the XX bin. XX are numbers, e.g. out.001.fasta
(out).summary	summary file describing which contigs are being classified into which bin.
(out).log	log file recording the core steps of MaxBin algorithm
(out).marker	marker gene presence numbers for each bin. This table is ready to be plotted by R or other 3rd-party software.
(out).marker.pdf	visualization of the marker gene presence numbers using R
(out).noclass	all sequences that pass the minimum length threshold but are not classified successfully.
(out).tooshort	all sequences that do not meet the minimum length threshold.

Now you can run a gene prediction on each genome bin and BLAST one sequence for each bin for a (very crude!) classification:

```
for i in max*fasta; do prodigal -p meta -a $i.genes.faa -d $i.genes.fna -f gff -o $i.
↪genes.gff -i $i& done
```

Does the abundance of the bins match the 16S profile of the community?

MetaBAT Binning

MetaBAT, An Efficient Tool for Accurately Reconstructing Single Genomes from Complex Microbial Communities.

Grouping large genomic fragments assembled from shotgun metagenomic sequences to deconvolute complex microbial communities, or metagenome binning, enables the study of individual organisms and their interactions. MetaBAT is an automated metagenome binning software which integrates empirical probabilistic distances of genome abundance and tetranucleotide frequency. See the MetaBAT home page for more info.

Let's run a MetaBAT binning on the MEGAHIT assembly:

```
cd ~/workdir/assembly/megahit_out

qsub -cwd -pe multislot 12 -N metabat -l mtc=1 -b y \
/vol/cmg/bin/runMetaBat.sh final.contigs.fa megahit_sorted.bam
```

MetaBAT will generate 12 bins from our assembly:

```
final.contigs.fa.metabat-bins-.1.fa
final.contigs.fa.metabat-bins-.2.fa
final.contigs.fa.metabat-bins-.3.fa
final.contigs.fa.metabat-bins-.4.fa
final.contigs.fa.metabat-bins-.5.fa
final.contigs.fa.metabat-bins-.6.fa
final.contigs.fa.metabat-bins-.7.fa
final.contigs.fa.metabat-bins-.8.fa
final.contigs.fa.metabat-bins-.9.fa
final.contigs.fa.metabat-bins-.10.fa
final.contigs.fa.metabat-bins-.11.fa
final.contigs.fa.metabat-bins-.12.fa
```

CHAPTER 7

Classification

Taxonomonic classification tools assign taxonommic labels to reads or assembled contigs of metagenomic datasets.

Kraken Taxonomic Sequence Classification System

Kraken is a system for assigning taxonomic labels to short DNA sequences, usually obtained through metagenomic studies. Kraken aims to achieve high sensitivity and high speed by utilizing exact alignments of k-mers and a novel classification algorithm.

In its fastest mode of operation, for a simulated metagenome of 100 bp reads, Kraken processed over 4 million reads per minute on a single core, over 900 times faster than Megablast and over 11 times faster than the abundance estimation program MetaPhlAn. Kraken's accuracy is comparable with Megablast, with slightly lower sensitivity and very high precision.

See the Kraken home page for more info.

Let's assign taxonomic labels to our binning results using Kraken. First, we need to compare the genome bins against the Kraken database:

```
cd ~/workdir/assembly/megahit_out

qsub -cwd -pe multislot 24 -N kraken -l mtc=1 -b y \
/vol/cmg/bin/kraken --db /vol/metagencourse/krakendb --threads 24 --fasta-input␣
↪maxbin.001.fasta --output maxbin.001.kraken
```

If you need the full taxonomic name associated with each input sequence, Kraken provides a script named kraken-translate that produces two different output formats for classified sequences. The script operates on the output of kraken:

```
kraken-translate --db /vol/metagencourse/krakendb maxbin.001.kraken > maxbin.001.
↪kraken.labels
```

Does the abundance of the bins match the 16S profile of the community?